

ΣΧΕΔΙΑΣΜΌΣ ΣΥΣΤΗΜΆΤΩΝ VLSI

Πρώτη Εργαστηριακή Άσκηση

Γώργος Ντάκος 1059569

17 Μαρτίου 2021

Περίληψη

Στο παρακάτω κείμενο παραδίδεται η αναφορά της πρώτης εργαστηριακής άσκησης του μαθήματος Σχεδιασμός Συστημάτων VLSI . Όλοι οι κώδικες έχουν συγγραφεί σε γλώσσα **VHDL** μέσω του **Notepad++** και έχουν μεταγλωττιστεί και επιβεβαιωθεί οι λειτουργίες τους μέσω του εργαλείου **ModelSim**.

Περιεχόμενα

1	Ενότητα Β	4
2	Ενότητα Γ	5
3	Κώδικες σε VHDL	11

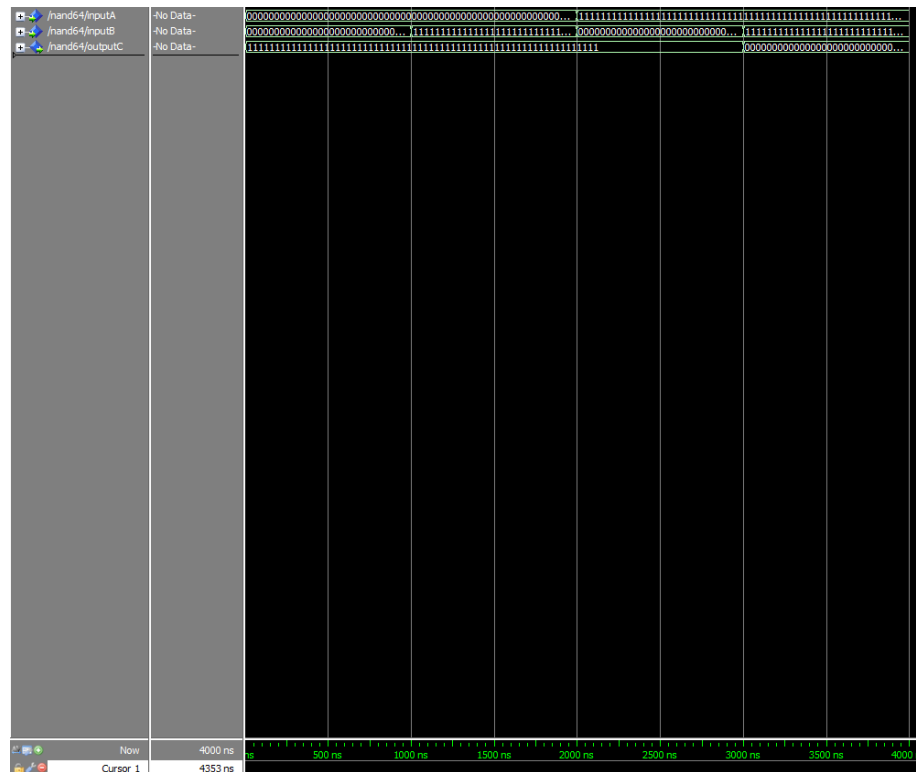
Κατάλογος Σχημάτων

1	Simulation της AND-64-bit	4
2	Πίνακας αληθείας του Full Adder	5
3	Simulation του Full Adder	5
4	Schematic του Full Adder	6
5	Full Adder	6
6	Πίνακες αληθείας AND & XOR.	7
7	Simulation του Logic-Unit	8
8	Logic-Unit	9
9	Πίνακας αληθείας DFF	9
10	Simulation του Logic-Unit-Reg	10
11	Logic-Unit-Reg	11

1 Ενότητα Β

B.1,B.2

Ο κώδικας του συγκεκριμένου υποερωτήματος αυτό που κάνει είναι να μου παράγει 64 διαφορετικές πύλες AND με την κάθε μία να έχει 2 εισόδους και 1 έξοδο. Ο πίνακας αληθείας της πύλης φαίνεται στην εικόνα 6. Αφού κάνουμε compile τον κώδικα τον εξομοιώνουμε και παίρνουμε τα εξής αποτελέσματα όπου και διαπιστώνουν την ορθή λειτουργία του:



Σχήμα 1: Simulation της AND-64-bit

B.3

Για το συγκεκριμένο ζητούμενο δεν γίνανε πολλές αλλαγές. Βασικά ο κορμός του κώδικα είναι ίδιος πέρα απο μια διαφορά και μια επιπλέον περιοχή στην οντότητα. Η επιπλέον περιοχή στην οντότητα με την ετικέτα **GENERIC** μας βοηθάει να τροποποιούμε πιο εύκολα το μέγεθος ενός δεδομένου όπως για εισόδους ή εξόδους. Τώρα η αλλαγή που κάναμε ήταν αντί για **downto** έχουμε βάλει τη λέξη **to** οπότε ανεβαίνουμε απο το 0 στο 63. Αυτή η αλλαγή βέβαια δεν μας βοηθάει και πολύ αφού μπορεί να μας μπερδέψει με το ποιό τεστφbit του τεστφvector είναι το τεστφMSB. Και στις 2 περιπτώσεις θα πρέπει να ξέρουμε ότι το τεστφMSB είναι το τεστφbit 63.

2 Ενότητα Γ

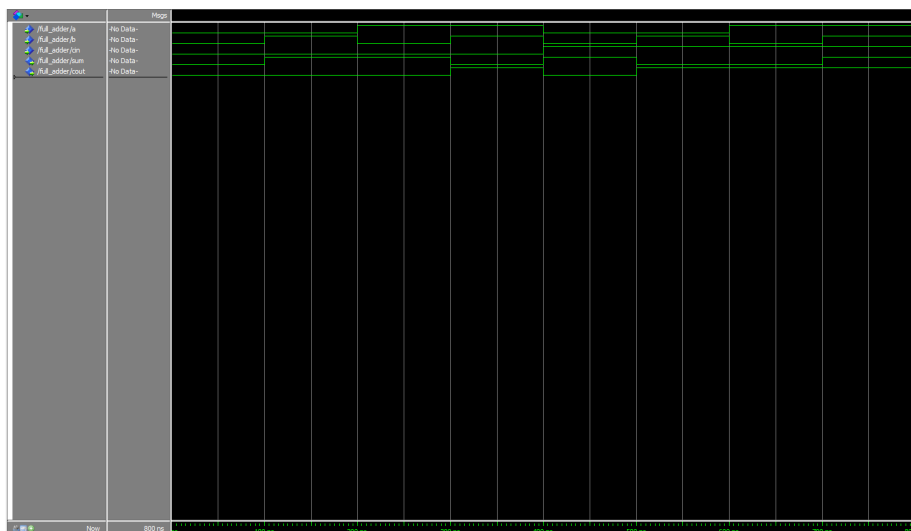
$\Gamma.1$

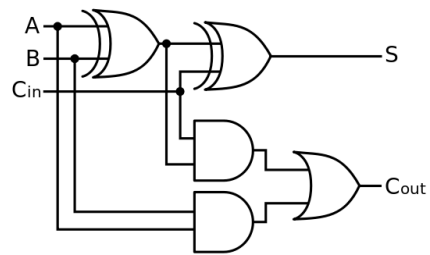
Για να φτιάξουμε το κύκλωμα του FA πρώτα πολλά δημιουργήσαμε το σχηματικό του έτσι ώστε να μπορούμε να εξάγουμε τον πίνακα αληθείας και να υλοποιήσουμε σε επίπεδο συναρτήσεων boole . Οπότε σύμφωνα με την εικόνα 4 ο πίνακας αληθείας είναι ο παρακάτω:

a	b	cin	sum	cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

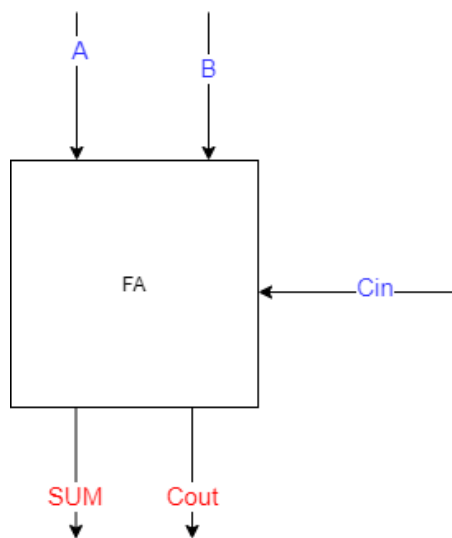
Σχήμα 2: Πίνακας αληθείας του Full Adder

Έπειτα από το compile του κώδικα και το simulate του καταλήγουμε στο ότι το κύκλωμα μας λειτουργεί ορθά όπως φαίνεται και στην εικόνα 3 όπου συμβαδίζει με αυτά του πίνακα αληθείας.





Σχήμα 4: Schematic του Full Adder



Σχήμα 5: Full Adder

Γ.2

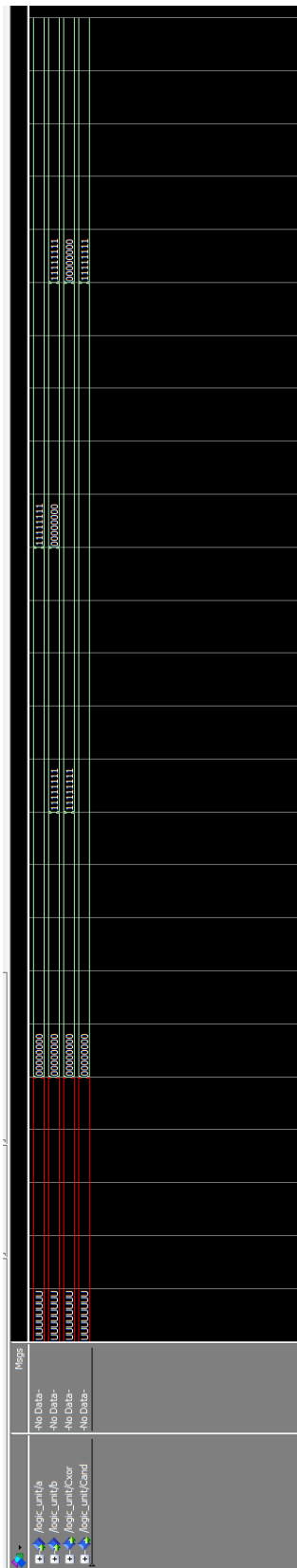
Για να φτιάξουμε το κύκλωμα που μας ζητάει πρώτα πόλλα δημιουργήσαμε το σχηματικό του έτσι ώστε να μπορέσουμε να εξάγουμε τον πίνακα αληθείας και να το υλοποιήσουμε σε επίπεδο συναρτήσεων boole . Οπότε σύμφωνα με την εικόνα 7 οι πίνακες αληθείας είναι οι παρακάτω:

Inputs			Outputs		
X	Y	Z	A	B	Output
0	0	0	0	0	0
0	1	1	0	1	0
1	0	1	1	0	0
1	1	0	1	1	1

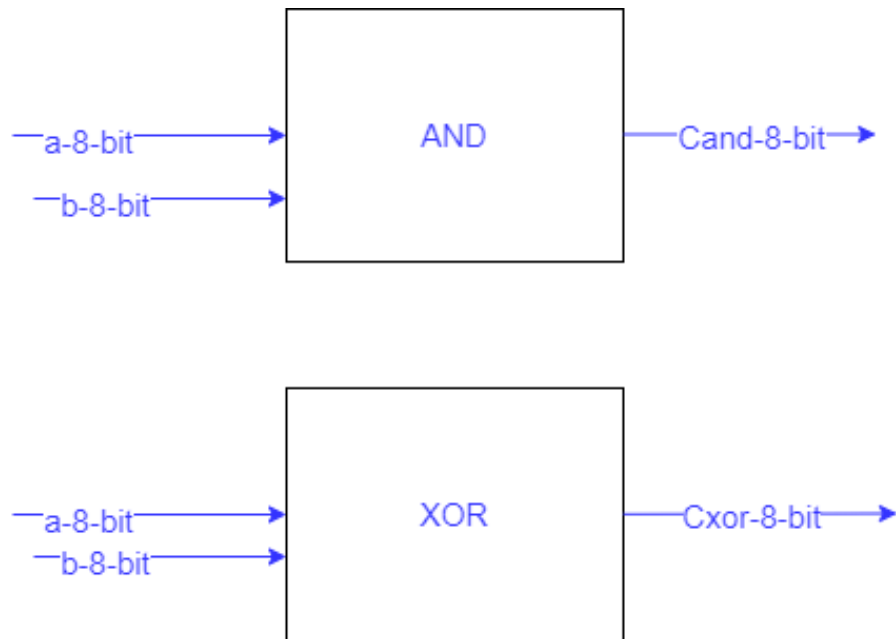
XOR GATE AND GATE

Σχήμα 6: Πίνακες αληθείας AND & XOR.

Έπειτα από το compile του κώδικα και το simulate του καταλήγουμε στο ότι το κύκλωμα μας λειτουργεί ορθά όπως φαίνεται και στην εικόνα 7 όπου συμβαδίζει με αυτά του πίνακα αληθείας.



Σχήμα 7: Simulation του Logic-Unit



Σχήμα 8: Logic-Unit

Γ.3

Το κυκλώμα που ζητάει το υποερωτήμα αυτό είναι να ενσωματώσουμε σε κάθε εξο-
δο του κυκλώματος της εικόνας 8 έναν καταχωρητή. Δηλαδή απλά θα προσθέσουμε
1 DFF για κάθε εξόδους του κυκλώματος όπως φαίνεται στην εικόνα 11. Παρακάτω
φαίνεται και ο πίνακας αληθείας του DFF :

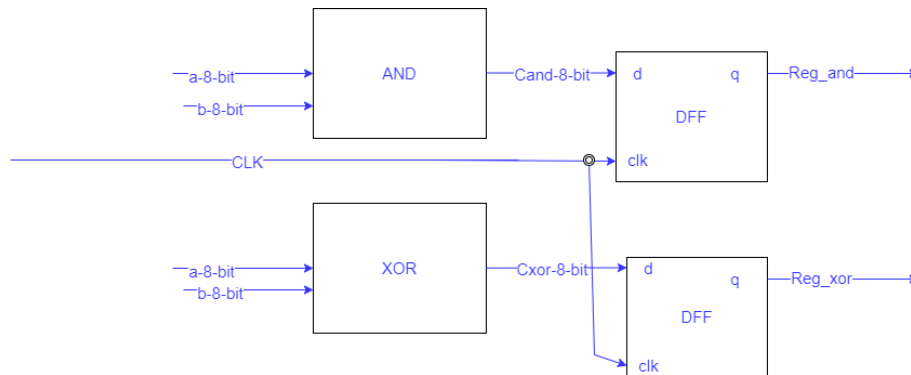
clk	D	Q
0	0	Q
0	1	Q
1	0	0
1	1	1

Σχήμα 9: Πίνακας αληθείας DFF

Έπειτα από το compile του κώδικα και το simulate του καταλήγουμε στο ότι το κύκλωμα μας λειτουργεί ορθά όπως φαίνεται και στην εικόνα 10 όπου συμβαδίζει με αυτά του πίνακα αληθείας.

Logic Unit, regA	Logic Unit, regB	Logic Unit, regC	Logic Unit, regD	Logic Unit, regQ3	Logic Unit, regQ2	Logic Unit, regQ1	Logic Unit, regQ0
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000001	00000000	00000000	00000000	00000001
00000000	00000000	00000001	00000000	00000000	00000001	00000000	00000000
00000000	00000000	00000001	00000001	00000000	00000001	00000001	00000000
00000000	00000001	00000000	00000000	00000001	00000000	00000000	00000000
00000000	00000001	00000000	00000001	00000001	00000000	00000001	00000000
00000000	00000001	00000001	00000000	00000001	00000001	00000000	00000000
00000000	00000001	00000001	00000001	00000001	00000001	00000001	00000000
00000001	00000000	00000000	00000000	00000001	00000000	00000000	00000000
00000001	00000000	00000000	00000001	00000001	00000000	00000001	00000000
00000001	00000000	00000001	00000000	00000001	00000001	00000000	00000000
00000001	00000000	00000001	00000001	00000001	00000001	00000001	00000000
00000001	00000001	00000000	00000000	00000001	00000001	00000000	00000000
00000001	00000001	00000000	00000001	00000001	00000001	00000001	00000000
00000001	00000001	00000001	00000000	00000001	00000001	00000001	00000001
00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001

Σχήμα 10: Simulation του Logic-Unit-Reg



Σχήμα 11: Logic-Unit-Reg

3 Κώδικες σε VHDL

Παρακάτω παρατίθενται οι κώδικες της εργασίας:

NAND-64

```
--Dhlwsh bibliothhkhhs kai tun paketun poy tha xrhsimopoihsoume
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
```

```
--Dhlwsh ontothtas
ENTITY nand64 IS
PORT (
  -- Shma eisodoy poy lambanei mia akoloythia timwn.
  inputA :IN std_logic_vector(63 downto 0);
  -- Shma eisodoy poy lambanei mia akoloythia timwn.
  inputB :IN std_logic_vector(63 downto 0);
  -- Shma ejodoy poy lambanei mia akoloythia timwn.
  outputC :OUT std_logic_vector(63 downto 0));
END nand64;
```

```
--Dhlwsh arxitektonikhhs
ARCHITECTURE structural OF nand64 IS
BEGIN
  -- Broxos o opoios tha ektelestei 64 fores.
  XORLoop: For i IN 0 TO 63 Generate
    --Se kathe epanalhpsh anathetoyme thn logikh prajh nand-
    --sto shma ejodoy ths kathe epanalhpshs.
    --Dhladh dhmiourgoyme 64 diaforetikes pyles nand.
    outputC(i) <= inputA(i) NAND inputB(i);
  End Generate XORLoop; --Telos Broxoy
```

~~-----NAND-64-Me-----~~

```
--Dhlwsh bibliothhkhhs kai tun paketun poy tha xrhsimopoihsoyme
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

--Dhlwsh ontothtas
ENTITY nand64me IS
  GENERIC(N: NATURAL :=64);
  PORT (
    -- Shma eisodoy poy lambanei mia akoloythia timwn.
    inputA :IN std_logic_vector(0 to N-1);
    -- Shma eisodoy poy lambanei mia akoloythia timwn.
    inputB :IN std_logic_vector(0 to N-1);
    -- Shma ejodoy poy lambanei mia akoloythia timwn.
    outputC :OUT std_logic_vector(0 to N-1));
  END nand64me;

--Dhlwsh arxitektonikhhs
ARCHITECTURE structural OF nand64me IS
  BEGIN
    -- Broxos o opoios tha ektelestei 64 fores.
    XORLoop: For i IN 0 TO 63 Generate
      --Se kathe epanalhps h anathetoyme thn logikh prajh nand-
      --sto shma ejodoy ths kathe epanalhps hs.
      --Dhladh dhmiourgoyme 64 diaforetikhs pyles nand.
      outputC(i) <= inputA(i) NAND inputB(i);
      End Generate XORLoop; --Telos Broxoy
    END structural;
```

~~—FULL ADDER—~~

```
--Dhlwsh bibliothhkhhs kai tun paketun poy tha xrhsimopoihsoyme
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

--Dhlwsh ontothtas
ENTITY full_adder IS
PORT
(
  -- Shmata eisodoy poy lambanoun 0 h 1.
  a,b,cin: IN BIT;
  -- Shmata ejodoy poy lambanoun 0 h 1.
  sum,cout: OUT BIT);
END full_adder;

--Dhlwsh arxitektonikhhs
ARCHITECTURE dataflow OF full_adder IS
  --sto parakatw mplok kwдика
  --orizoyne thn leitourgia ths
  --ontothtas me perigrafh rohs dedomenwn
BEGIN
  sum <= a XOR b XOR cin;
  cout <=(a AND b) OR (a AND cin) OR (b AND cin);
END dataflow;
```

-----Logic-Unit-----

```
--Dhlwsh bibliothhkhhs kai tun paketun poy tha xrhsimopoihsoyme
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

--Dhlwsh ontothtas
ENTITY logic_unit IS
PORT
(
  -- Shmata eisodoy poy lambanoun mia akoloythia timwn.
  a,b: IN STD_LOGIC_VECTOR(7 downto 0);
  -- Shmata ejodoy poy lambanoun mia akoloythia timwn.
  Cxor,Cand: OUT STD_LOGIC_VECTOR(7 downto 0));
END logic_unit;

--Dhlwsh arxitektonikhhs
ARCHITECTURE dataflow OF logic_unit IS
  --sto parakatw mplok kwдика
  --orizoyme thn leitourgia ths
  --ontothtas me perigrafh rohs dedomenwn
  BEGIN
    ForLoop: For i IN 0 TO 7 Generate
      --Se kathe epanalhpsh anathetoyme thn logikh prajh and & xor
      --sta shmata ejodoy antistoixa ths kathe epanalhpshs.
      --Dhladh dhmiourgoyme 8 diaforetikes pyles and
      --kai 8 diaforetikes pyles xor
      Cand(i) <= a(i) AND b(i);
      Cxor(i) <= a(i) XOR b(i);
    End Generate ForLoop; --Telos Broxoy
  END dataflow;
```

~~-----Logic-Unit-REG-----~~

```
--Dhlwsh bibliothhkhhs kai tun paketun poy tha xrhsimopoihsoyme
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

--Dhlwsh ontothtas
ENTITY logic_unit_reg IS
PORT
(
clk: IN STD_LOGIC;
-- Shmata eisodoy poy lambanoun mia akoloythia timwn.
a,b: IN STD_LOGIC_VECTOR(7 downto 0);
-- Shmata ejodoy poy lambanoun mia akoloythia timwn.
Reg_xor,Reg_and: OUT STD_LOGIC_VECTOR(7 downto 0));
END logic_unit_reg;

--Dhlwsh arxitektonikhhs
ARCHITECTURE dataflow OF logic_unit_reg IS
SIGNAL Cxor,Cand: STD_LOGIC_VECTOR(7 downto 0);
--sto parakatw mplok kwdika
--orizoyme thn leitourgia ths
--ontothtas me perigrafh rohs dedomenwn
BEGIN
ForLoop: For i IN 0 TO 7 Generate
--Se kathe epanalhpsh anathetoyme thn logikh prajh and & xor
--sta shmata ejodoy antistoixa ths kathe epanalhpshs.
--Dhladh dhmiourgoyme 8 diaforetikes pyles and
--kai 8 diaforetikes pyles xor
Cand(i) <= a(i) AND b(i);
Cxor(i) <= a(i) XOR b(i);
--Akolouthiakhs diadikasias
PROCESS(clk)
BEGIN
IF
(clk'EVENT AND clk='1') THEN
Reg_xor(i) <= Cxor(i);
Reg_and(i) <= Cand(i);
END IF;
END PROCESS;
End Generate ForLoop; --Telos Broxoy
END dataflow;
```